TD #3 — Correction (avec solutions)

Le but de ce TD est de prouver la correction d'un certain nombre d'algorithmes (écrits en C) plus ou moins complexes. Les preuves de correction pourront utiliser des invariants *de boucle* qui satisfont les conditions suivantes : $\mathcal{I}$ est un *invariant (utile) de la boucle* `while (cond) { ... }` si :

— $\mathcal{I}$ est vrai avant la première itération de la boucle
— Si $\mathcal{I}$ et la condition d'entrée dans la boucle sont vrais, alors $\mathcal{I}$ est vrai à la fin d'une itération du corps de boucle *si cette itération termine*

Si la boucle ne contient pas de `break`, ceci permet de déduire que si l'on atteint le point du programme suivant immédiatement la boucle, alors $\mathcal{I}$ et la négation de `cond` y sont vrais.
Un (tout petit) peu plus formellement, soit `S` la suite d'instruction d'un corps de boucle `while` *sans* `break`, si l'on a la propriété : « si $\mathcal{I} \wedge \mathcal{C}$ sont vrais avant d'exécuter `S`, alors $\mathcal{I}$ est vrai après exécution de `S` », alors si $\mathcal{I}$ est vrai avant `while (C) S`, on a $\mathcal{I} \wedge \neg\mathcal{C}$ après l'exécution de la boucle toute entière.

Exercice 1. *Élément maximum dans un tableau*

On se donne la fonction C suivante :

```c
int max(size_t n, int a[n]) {
    assert(n > 0); // requis par le langage
    int m = a[0];
    size_t i = 1; // pour simplifier la rédaction
                  // de l'invariant
    for (; i < n; i++)
    {
        if (a[i] > m) {
            m = a[i];
        }
    }
    return m;
}
```

1. Prouvez sa terminaison.

   Cette fonction comporte une unique boucle `for` qui effectue exactement $n-1$ itérations : elle termine trivialement.

2. Prouvez sa correction relativement aux spécifications informelles suivantes : « `max` renvoie un élément maximum du tableau `a`, c'est à dire un élément `m` présent dans `a` et tel qu'il n'existe aucun autre élément `x` de `a` avec `x > m` ».

   Dans tout ce qui suit, on note m, m' & i, i' les valeurs des variables `m` & `i` atteintes en début et fin d'une itération de la boucle.
   On va prouver la correction en utilisant l'invariant de boucle :
   $\mathcal{I}$: m est un élément de `a` maximal parmi ceux d'indices strictement inférieurs à i, et $0 < i \leq n$.

   Initialisation : Avant la première itération de la boucle, m est égal à `m[0]` et $i = 1$, donc $\mathcal{I}$ est vrai.

   Conservation : On suppose $\mathcal{I}$ vrai en début d'une itération. Par la condition de boucle et $\mathcal{I}$, $0 < i < n$ et $a[i]$ est un élément du tableau `a`. Par définition du compteur de boucle on a $i' = i + 1 \leq n$. Par l'instruction `if` dans le corps de boucle m' est égal au maximum entre m et $a[i]$, ce qui par $\mathcal{I}$ est égal à $\max(\max_{0 \leq j < i} a[j], a[i])$, soit $\max_{0 \leq j \leq i} a[j] = \max_{0 \leq j < i'} a[j]$ par les propriétés du maximum. Mis tout ensemble, $\mathcal{I}$ reste vrai à la fin de l'itération.

   On peut maintenant conclure en observant que la valeur renvoyée par `max` est égale à la valeur m atteinte par `m` immédiatement en sortie de boucle. Alors, par la condition de sortie de celle-ci l'on a $i \geq n$, et par $\mathcal{I}$ l'on a $i \leq n$, donc $i = n$ et (toujours par $\mathcal{I}$) $m' = \max_{0 \leq j < n} a[i]$ comme voulu.

Exercice 2. *Recherche d'un élément dans un tableau*

On se donne la fonction C suivante :

```
1  int search(int n, int a[n], int e) {
2      assert(n > 0); // requis par le langage
3      int i = 0; // pour simplifier la rédaction
4                 // de l'invariant
5      for (; i < n; i++)
6      {
7          if (a[i] == e) {
8              return i;
9          }
10     }
11     return -1;
12 }
```

1. Prouvez sa terminaison.

> Cette fonction comporte une unique boucle `for` qui effectue au plus n itérations : elle termine trivialement.

2. Prouvez sa correction relativement aux spécifications informelles suivantes : « `search` renvoie `-1` si aucun élément de `a` n'est égal à `e`, et sinon renvoie une valeur `i` telle que `a[i] == e` ».

> Dans tout ce qui suit, on note i, i' & e les valeurs des variables `i` en début et fin d'itération de boucle, & de `e` tout au long de l'exécution.
> On va prouver la correction en utilisant l'invariant de boucle :
> $\mathcal{I}$: $0 \leq i \leq n$ et il n'existe aucun élément de a d'indice $< i$ égal à e.
>
> Initialisation : Avant la première itération, par l'initialisation de `i` et l'`assert` en ligne 2, $0 \leq i = 0 \leq n$ et il n'existe trivialement aucun élément de a d'indice < 0 égal à e : $\mathcal{I}$ est vrai.
>
> Conservation : On suppose $\mathcal{I}$ vrai en début d'une itération. Par la condition de boucle et $\mathcal{I}$, $0 \leq i < n$, et $a[i]$ est un élément du tableau `a`, et par définition du compteur de boucle $i' = i + 1 \leq n$.
> Par l'instruction `if` du corps de boucle, `search` termine son exécution si $a[i] = e$, ce qui est correct pour ses spécifications. Si la fin de l'itération est atteinte cela veut dire que $a[i] \neq e$; par $\mathcal{I}$ e n'est égal à aucun élément de `a` d'indice $< i$, donc aucun élément de a d'indice $\leq i$, ou $< i'$. Mis tout ensemble, $\mathcal{I}$ reste vrai en fin d'exécution.
>
> On peut maintenant terminer la preuve en considérant les deux points de retour possible de `search`. On a déjà établi ci-dessus que celui de la ligne 6 était correct (par $\mathcal{I}$ et les conditions du test le déclenchant), et il ne reste qu'à analyser celui de la ligne 9. Ce dernier se trouvant immédiatement après la fin de la boucle, on a par la condition de sortie de cette dernière que $i \geq n$. On a aussi par $\mathcal{I}$ que $i \leq n$ donc $i = n$. Enfin on a toujours par $\mathcal{I}$ qu'il n'existe aucun élément de `a` d'indice $< n$ égal à e, et il est donc correct de renvoyer `-1`.

Exercice 3. *Recherche d'un élément dans un tableau trié, par dichotomie* ⋆⋆

On se donne la fonction C suivante :

```
1  int bsearch(int n, int a[n], int e) {
2      int b = 0;
3      int t = n - 1;
4
5      while (b < t) {
6          int m = b + (t - b)/2;
7          if (e == a[m]) {
8              return m;
9          }
10         if (e < a[m]) {
11             t = m - 1;
12         }
13         else {
14             b = m + 1;
15         }
16     }
17
18     if ((b > t) || (a[b] != e)) {
```

```
19            return -1;
20        }
21        return b;
22    }
```

Dans tout ce qui suit, on note b, t, b', t' les valeurs des variables `b` et `t` atteintes en début et (éventuelle) fin d'une itération de la boucle, respectivement. On note également m l'unique valeur de la variable `m` au cours d'une itération.

1. Prouvez sa terminaison

La terminaison de cette fonction n'est pas complètement évidente puisqu'elle contient une (unique) boucle `while` dont le nombre d'itérations ne peut pas être immédiatement borné. On peut néanmoins prouver sa terminaison en trouvant un variant, et l'on va montrer que $t-b$ en est un :

Minoration : la condition d'entrée de boucle fait qu'à chaque début d'itération $t-b>0$; $t-b$ est donc minoré par une constante.

Décroissance : On veut montrer que $t'-b'<t-b$; lors d'une itération trois cas sont possible :

- On n'atteint pas la fin du corps de boucle car le test effectué par le premier `if` s'évalue à `true` ; dans ce cas la fonction toute entière termine et il n'y a plus rien à prouver.
- $t'=t$, $b'=m+1$. Par définition de m et le fait que $t-b>0$, on a $m\geq b$, et donc $m+1>b$. Il s'ensuit que $t'-b'=t-(m+1)<t-b$.
- $t'=m-1$, $b'=b$. On a $t'-b'=m-1-b$, ce qui par définition de m peut s'écrire $b+(t-b)\div 2-1-b=(t-b)\div 2-1<t-b$, où la dernière inégalité vient du fait que $(t-b)\div 2\leq t-b$, puisque $(t-b)$ est positif.

Il s'ensuit que $t-b$ est un variant, et que la boucle (et donc la fonction) termine.

2. Prouvez sa correction relativement aux spécifications informelles suivantes : « soit `a` un tableau d'éléments triés par ordre croissant, `bsearch` renvoie `-1` si aucun élément de `a` n'est égal à `e`, et sinon renvoie une valeur `i` telle que `a[i] == e` ».

On va prouver la correction grâce à deux invariants pour la boucle `while`.
On commence par :
$\mathcal{I}_1$: $b\geq 0$ et $t<n$.

Initialisation : On veut montrer que $\mathcal{I}_1$ est vrai avant la première itération de la boucle. À la ligne 4, $b=0$, $t=n-1$, et $\mathcal{I}_1$ est vrai.

Conservation : On suppose $\mathcal{I}_1$ vrai en début d'une itération.
Par la condition de boucle on a $b<t$, et donc $m=b+(t-b)\div 2\geq b$. Les instructions du corps de boucle sont telles que l'on a $b'=b$ ou $b'=m+1>b$ en fonction de si la ligne 14 est exécutée, ce qui par $\mathcal{I}_1$ est positif dans les deux cas.
De même on a $t'=t$ ou $t'=m-1$ en fonction de si la ligne 11 est exécutée ; or on a déjà prouvé plus haut que $m-1-b<t-b$, d'où $t'\leq t$ dans tous les cas, et donc $t'<n$ par $\mathcal{I}_1$.
$\mathcal{I}_1$ est donc vrai en fin d'itération.

On montre ensuite :
$\mathcal{I}_2$: Si `e` est présent dans `a`, il ne peut l'être qu'à un indice $\in[\![b,t]\!]$.

Initialisation : À la ligne 4, $b=0$, $t=n-1$: l'intervalle $[\![b,t]\!]$ correspond alors exactement à celui des indices valides d'éléments de `a`, et $\mathcal{I}_2$ est vrai.

Conservation :
On suppose $\mathcal{I}_2$ vrai en début d'une itération. Si la fin de l'itération est atteinte on a deux cas possibles pour `b` et `t` : $b'=b$ et $t'=m-1$, ou $b'=m+1$ et $t'=t$.
Par les conditions du `if` de la ligne 7 et de celui de la ligne 10, on a que le premier cas correspond au fait que `a[m]` est strictement supérieur à `e`. Or le tableau `a` étant trié, cela veut dire que si `e` est présent dans `a`, il l'est à un indice $\in[\![0,m-1]\!]$. Par $\mathcal{I}_2$, il ne peut aussi l'être qu'à un indice $\in[\![b,t]\!]$, et donc ne peut l'être qu'à un indice $[\![0,m-1]\!]\cap[\![b,t]\!]$, ce qui par $\mathcal{I}_1$ et $t'\leq t$ est égal à $[\![b',t']\!]$.
Par les conditions des mêmes `if`s, le second cas correspond au fait que `a[m]` est strictement inférieur à `e`. Par les mêmes arguments que ci-dessus, si `e` est présent dans `a` il ne peut l'être qu'à un indice $\in[\![m+1,n-1]\!]\cap[\![b,t]\!]$, ce qui par $\mathcal{I}_1$ et $m+1>b$ est égal à $[\![b',t']\!]$.
$\mathcal{I}_2$ est donc vrai en fin d'itération.

On peut maintenant conclure en considérant les trois points de fin d'exécution possibles de la fonction :

- L'unique `return` à l'intérieur de la boucle : par $\mathcal{I}_1$ et la condition du test déclenchant ce `return`, `a[m]` est un élément valide de `a` égal à `e`, et il est donc correct de renvoyer m.

— Les deux `return` après la fin de la boucle. On note toujours b et t les valeurs des variables `b` et `t`. Puisque l'exécution de la boucle a terminé, cela veut dire que $b \geq t$, et par $\mathcal{I}_1$ et $\mathcal{I}_2$ on a également $0 \leq b$ et $t < n$ et que si `e` est présent dans `a` il ne peut l'être qu'à un indice $[\![b, t]\!]$. Si $b > t$ cet intervalle est vide, et `e` n'est pas présent dans `a` : il est correct de renvoyer `-1` ; sinon $b = t$ désigne un indice valide de l'unique élément dans `a` pouvant être égal à `e`, et il est correct de renvoyer `-1` si cet élément ne lui est pas égal, et b sinon.

Exercice 4. *Exponentiation lente*

Prouvez la terminaison et la correction de votre fonction `sloexp` du TD#2.

On définit :

```
1  uint64_t sloexp(uint64_t a, uint64_t b)
2  {
3      uint64_t r = 1;
4      uint64_t i = 0;
5
6      for (; i < b; i++)
7      {
8          r = r * a;
9      }
10
11     return r;
12 }
```

qui déclare son compteur de boucle en dehors de celle-ci afin de faciliter la rédaction de l'invariant de boucle.

La terminaison de l'unique boucle `for`, et donc du programme est évidente.

On note a, b, les valeurs (constantes) prises par les variables `a` et `b`, et r et r' & i et i' celles prises par les variables `r` & `i` en début et fin d'itération de la boucle `for`. Les spécifications de `sloexp` imposent qu'elle doive renvoyer une valeur égale à a^b à condition que ce nombre soit représentable par une valeur de type `uint64_t`, et elle est de comportement non défini sinon. On peut donc supposer ci-dessous que les résultats de tous les calculs sont représentables par une valeur de type `uint64_t`, car il n'y a rien à prouver sinon.
Pour prouver la correction, on utilise l'invariant de boucle suivant :
$\mathcal{I}$: $r = a^i$ et $i \leq b$

Initialisation : À la ligne 5 on a $r = 1$ et $i = 0$ donc $r = a^i$. De plus $b \geq 0$ de par le typage de `b`, donc $i \leq b$, et $\mathcal{I}$ est vrai.

Conservation : On suppose $\mathcal{I}$ vrai en début d'une itération. Par la condition de boucle on a $i < b$; par l'expression d'incrément `i++` utilisée dans la boucle `for` on a $i' = i + 1$, et il s'ensuit que $i' \leq b$ à la fin de l'itération.
Par la ligne 8 on a $r' = r \times a$; par $\mathcal{I}$ c'est aussi $r' = a^i \times a$, d'où $r' = a^{i+1} = a^{i'}$.
$\mathcal{I}$ est bien vrai en fin d'itération.

Pour conclure, il suffit d'observer que la valeur renvoyée par `sloexp` est la valeur r' atteinte par `r` immédiatement en sortie de boucle. Alors, par la condition de sortie de boucle on a $i' \geq b$, et par $\mathcal{I}$ on a $i' \leq b$ et $r' = a^{i'}$, d'où $r' = a^b$.

Exercice 5. *Exponentiation rapide*

Prouvez la terminaison et la correction de votre fonction `fastexp` du TD#2.

On définit :

```
1  uint64_t fastexp(uint64_t a, uint64_t b)
2  {
3      uint64_t r = 1;
4
5      for ( ; b > 0 ; b /= 2)
6      {
7          if (b % 2 == 1)
8          {
9              r = r * a;
10         }
11
12         a = a * a;
13     }
```

```
14
15      return r;
16  }
```

La terminaison de l'unique boucle `for`, et donc du programme est évidente.

On note α et β les valeurs initiales des arguments `a` et `b`, puis a et a', b et b', r et r' les valeurs prises par les variables `a`, `b` et `r` en début et fin d'itération de la boucle `for`. Les spécifications de `fastexp` imposent qu'elle doive renvoyer une valeur égale à α^β à condition que ce nombre soit représentable par une valeur de type `uint64_t`, et elle est de comportement non défini sinon. On peut donc supposer ci-dessous que les résultats de tous les calculs sont représentables par une valeur de type `uint64_t`, car il n'y a rien à prouver sinon.
Pour prouver la correction, on utilise l'invariant de boucle suivant :
$\mathcal{I} : r \times a^b = \alpha^\beta$

Initialisation : À la ligne 4 on a $r = 1$, $a = \alpha$, $b = \beta$, et $\mathcal{I}$ est vrai.

Conservation : On suppose $\mathcal{I}$ vrai en début d'une itération. On distingue deux cas : $b = 2b' + 1$ est impair, et $b = 2b'$ est pair. Par la condition du `if` de la ligne 7 on a dans le premier cas que $r' = r \times a$, et dans le second que $r' = r$. Dans tous les cas, $a' = a^2$. Enfin par $\mathcal{I}$ on a $r \times a^b = \alpha^\beta$.
Cas b impair : $r' \times a'^{b'} = r \times a \times (a^2)^{b'} = r \times a \times a^{2b'} = r \times a^{2b'+1} = r \times a^b = \alpha^\beta$
Cas b pair : $r' \times a'^{b'} = r \times (a^2)^{b'} = r \times a^{2b'} = r \times a^b = \alpha^\beta$

Pour conclure, il suffit d'observer que la valeur renvoyée par `fastexp` est la valeur r' atteinte par `r` en sortie de boucle. Par la condition de celle-ci, cela implique $b' = 0$, d'où par $\mathcal{I}$ $r' \times (a')^0 = \alpha^\beta$, et donc $r' = \alpha^\beta$.

Teaser : prouvez que la fonction ci-dessous calcule le PGCD de a et b, quand $a \geq b$.

```
uint64_t bgcd(uint64_t a, uint64_t b) {
    uint64_t r = 1;
    while (a > 0 && b > 0) {
        while ((a % 2 == 0) && (b % 2) == 0) {
            r *= 2;
            a /= 2;
            b /= 2;
        }
        while (a % 2 == 0) {
            a /= 2;
        }
        while (b % 2 == 0) {
            b /= 2;
        }
        if (a < b) {
            b = b - a;
        }
        else {
            uint64_t t = b;
            b = a - b;
            a = t;
        }
    }
    return a * r;
}
```